APPLICATION FOR UNITED STATES LETTERS PATENT

For

# METHOD FOR PROVIDING GARBAGE COLLECTION SUPPORT

Inventors:

Tatiana Shpeisman

Guei-Yuan Lueh

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
32400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(408) 720-8598

Attorney's Docket No.: 42390P11915

# METHOD FOR PROVIDING GARBAGE COLLECTION SUPPORT

## FIELD OF THE INVENTION

**[0001]** This invention relates generally to computing system memory management, and more specifically, to an efficient method for providing garbage collection support.

## BACKGROUND OF THE INVENTION

**[0002]** The random access memory (RAM) of a computing system is typically a fixed-size resource. The RAM must be managed properly to maintain system performance. In run-time environments (e.g., JAVA) the memory management is handled by the system. Memory management includes a process known as garbage collection, which is a process of recycling memory. When a computing program is running it allocates and uses portions of memory on an on-going basis. At some point the program may no longer need to use a particular portion of memory, for example, the memory was allocated for a particular purpose that is no longer relevant. The portions that are no longer being used (garbage) are identified (collected) so that they can be recycled for future use. When garbage collection begins, the garbage collection algorithm must determine all live objects. That is, the garbage collection process marks all objects that are currently being used by the program (live references) and all other objects are considered garbage. The garbage collector (GC) relies on the system compiler to provide a live reference enumeration root set, i.e., live reference information. To provide this information to the GC, the compiler must keep track of which objects should be added to, or deleted from, the live reference information. The live reference information is constantly changing because at different points of program execution some

objects are being created and some are no longer used. Typically garbage collection occurs at the time of a function call (call) that requests new memory. A function call passes control to a subroutine and after the subroutine is executed, control returns to the next instruction in the main program. Therefore, to provide the live reference information to the GC, the compiler should be able to determine the live references at each call site. Typically, the compiler stores the live reference information by keeping a list of call sites with corresponding live reference information for each call site stored somewhere in memory. At compilation time, the compiler generates native code to be executed by the computing system and also stores a garbage collection map.

**[0003]** **Figure 1** illustrates a typical storage scheme for live reference information in accordance with the prior art. System 100, shown in **Figure 1**, includes native code storage 105 and garbage collection information storage 110. Garbage collection information storage 110 includes method garbage collection information storage 111 that contains a list of all variables for the program and live reference information storage 112. The program contained in native code section 105 consists of a set of instructions, some of which are calls. When the program is being executed and a request for new memory is made (e.g., at a call), the GC inquires of the compiler for a list of live references at that point in the program. So, for example, native code storage 105 contains calls A, B, and C. When garbage collection takes place at one of these calls, the address of the call site at which garbage collection happened is provided to the compiler. The compiler must then go to live reference information storage 112 of garbage collection information storage 110 and traverse the list until the specific address is found. That is, the compiler has to search through the list of all call sites until it finds the one where garbage

collection has occurred. At this point the GC has access to the live reference information to perform the garbage collection function.

**[0004]** Storing this information separately taxes the system's processing resources because the live reference information is obtained through a search of what may be a large list. The size of the list depends upon the number of calls in the method. Typically, the list may be indexed to reduce the information access time.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0005]** The present invention is illustrated by way of example, and not limitation, by the figures of the accompanying drawings in which like references indicate similar elements and in which:

**[0006]** **Figure 1** illustrates a typical storage scheme for live reference information in accordance with the prior art;

**[0007]** **Figure 2** is a diagram illustrating an exemplary computing system 200 for implementing compiler support for garbage collection in accordance with the present invention;

**[0008]** **Figure 3** illustrates a storage scheme for live reference information in accordance with the one embodiment of the present invention; and

**[0009]** **Figure 4** is a process flow diagram in accordance with one embodiment of the present invention.

## DETAILED DESCRIPTION

**[0010]** An improved method for compiler support of garbage collection is described. A runtime computing system may contain null operation instructions (NOPs) that contain a data field that is ignored by the hardware. For example, Intel Corporation's IA-64 architecture specifies that instructions be placed in groups of three instructions called bundles. The architecture also specifies which instructions may be bundled together. Empty slots within a bundle caused by the architecture specifications are filled with NOPs. In one embodiment, the NOPs are used to store the live reference data used by the GC. This allows the live reference information to be encoded right after the call instruction and thus the information can be retrieved in constant time. During compilation, the compiler knows which references are live at each call. The compiler may also determine the number of bits that may be used to store those references. In one embodiment the number of NOPs is sufficient to store the information. In an alternative embodiment, NOPs are artificially inserted to store the information.

**[0011]** **Figure 2** is a diagram illustrating an exemplary computing system 200 for implementing compiler support for garbage collection in accordance with the present invention. The live reference data generation, data storage calculations, NOP insertions, and information recordation described herein can be implemented and utilized within computing system 200, which can represent a general-purpose computer, portable computer, or other like device. The components of computing system 200 are exemplary in which one or more components can be omitted or added. For example, one or more memory devices can be utilized for computing system 200.

**[0012]** Referring to **Figure 2**, computing system 200 includes a central processing unit 202 and a signal processor 203 coupled to a display circuit 205, main memory 204, static memory 206, and mass storage device 207 via bus 201. Computing system 200 can also be coupled to a display 221, keypad input 222, cursor control 223, hard copy device 224, input/output (I/O) devices 225, and audio/speech device 226 via bus 201.

**[0013]** Bus 201 is a standard system bus for communicating information and signals. CPU 202 and signal processor 203 are processing units for computing system 200. CPU 202 or signal processor 203 or both can be used to process information and/or signals for computing system 200. CPU 202 includes a control unit 231, an arithmetic logic unit (ALU) 232, and several registers 233, which are used to process information and signals. Signal processor 203 can also include similar components as CPU 202.

**[0014]** Main memory 204 can be, e.g., a random access memory (RAM) or some other dynamic storage device, for storing information or instructions (program code), which are used by CPU 202 or signal processor 203. Main memory 204 may store temporary variables or other intermediate information during execution of instructions by CPU 202 or signal processor 203. Static memory 206, can be, e.g., a read only memory (ROM) and/or other static storage devices, for storing information or instructions, which can also be used by CPU 202 or signal processor 203. Mass storage device 207 can be, e.g., a hard or floppy disk drive or optical disk drive, for storing information or instructions for computing system 200.

**[0015]** Display 221 can be, e.g., a cathode ray tube (CRT) or liquid crystal display (LCD). Display device 221 displays information or graphics to a user. Computing system 200 can interface with display 221 via display circuit 205. Keypad input 222 is a

alphanumeric input device with an analog to digital converter. Cursor control 223 can be, e.g., a mouse, a trackball, or cursor direction keys, for controlling movement of an object on display 221. Hard copy device 224 can be, e.g., a laser printer, for printing information on paper, film, or some other like medium. A number of input/output devices 225 can be coupled to computing system 200. Compiler support for garbage collection in accordance with the present invention can be implemented by hardware and/or software contained within computing system 200. For example, CPU 202 or signal processor 203 can execute code or instructions stored in a machine-readable medium, e.g., main memory 204.

**[0016]** The machine-readable medium may include a mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine such as computer or digital processing device. For example, a machine-readable medium may include a read only memory (ROM), random access memory (RAM), magnetic disk storage media, optical storage media, flash memory devices. The code or instructions may be represented by carrier-wave signals, infrared signals, digital signals, and by other like signals.

**[0017]** **Figure 3** illustrates a storage scheme for live reference information in accordance with the one embodiment of the present invention. The system 300, shown in Figure 3, includes native code 305 containing, among other code, NOPs that store the live references information pertaining to each call in the program.

**[0018]** System 300 also contains garbage collection information 310, which is a list of all the variables in the program that contain live references during some point of program execution. In one embodiment the live references are reported by a number

corresponding to the list contained in the garbage collection information 310. For a program with four variables A, B, C, and D, the variables that are alive at a given call may be represented by a four-bit number. Only one such mapping of variables to numbers is required per method. When live variables are reported, the bit vector may be used with 1s (ones) in positions to correspond to live variables. For example if variables A, B, and D are live, at a given call, this fact may be represented by 1101 where a 1 (one) indicates the variable is alive and a 0 (zero) indicates the variable is not alive. This is in contrast to encoding the actual register for each reference, which for systems having 128 registers, for example, would require seven bits to store each live reference.

**[0019]** Using such bit-vector representation, and a NOP having a 21-bit field allows reporting on up to 21 objects using one NOP. This saves storage space because in some typical architectures, up to one third of the instructions are NOPs.

**[0020]** **Figure 4** is a process flow diagram in accordance with one embodiment of the present invention. Process 400, shown in **Figure 4**, begins with operation 405 in which the compiler generates live reference information. At operation 410, the number of bits required to record the live reference information is computed for each call site. For a given method, this number may be equal to the number of registers and stack location used by the method that may contain references as discussed above.

**[0021]** At operation 415, the number of NOP instructions required to store the live reference information is determined by dividing the number of bits calculated at operation 410 (equal to possible references of the method) by the number of bits in the NOP. For some architectures a NOP may contain a 21-bit field. Therefore, for methods with up to 21 registers and stack locations that may contain references, only one NOP is

required. Empirically it is found that for a majority of programs one NOP is sufficient. Two NOPs may be required in rare cases (i.e., those having more than 21 and up to 42 possible references). For a system architecture having 128 registers, the theoretical maximum number of NOPs required would be seven.

**[0022]** At operation 420 the required number of NOPs, as determined at operation 415, are inserted before each call. This may be done in alternative ways including either prior to, or during, templatization (the process of grouping instructions into bundles). As discussed above, instructions are grouped into bundles of three instructions. Due to a set of restriction as to which instructions may be bundled together, approximately one third of all instructions are NOPs. In one embodiment the required NOPs are inserted prior to templatization. Prior to templatization the list of useful instructions is reviewed and an appropriate number of NOPs are inserted just before each call instruction. In most cases this will be just one NOP instruction. Because there was likely to have been at least one NOP inserted anyway, due to bundling restrictions, the overall size of the bundled code is not increased significantly. In an alternative embodiment, the bundled instructions are reviewed to determine if NOPs are available where necessary (i.e., prior to a call). In those places where a NOP is required, but does not happen to be available, a NOP is inserted. Typically no more than one NOP will need to be inserted.

**[0023]** The live reference information is then recorded into each of the inserted NOPs at operation 425. These may be NOPs that were specifically inserted prior to templatization, or NOPs that occurred during the templatization process. In any case, a sufficient number of NOPs is available so that the live references information for each call may be inserted.

When garbage collection occurs, the method garbage collection information 310 is read to indicate how many instructions were used to record the live reference information. The required number of instructions are then passed with the call and the live reference information for the call is obtained by the GC.

**[0024]** In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense.